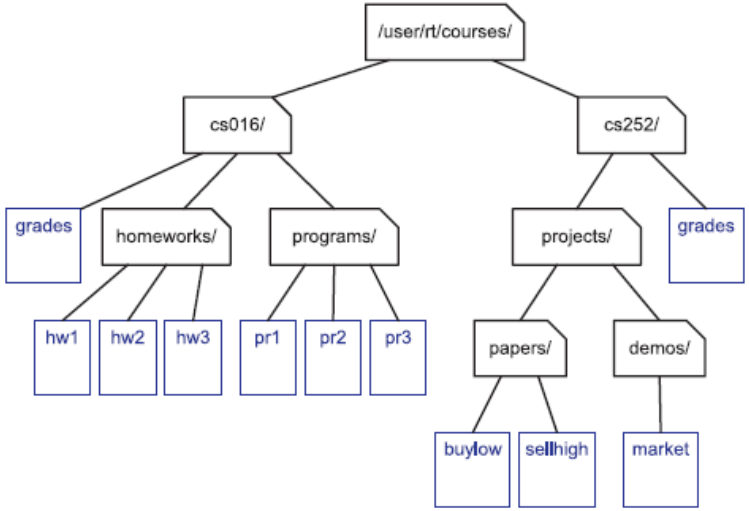
트리

0 목차

1 일반 트리 정의

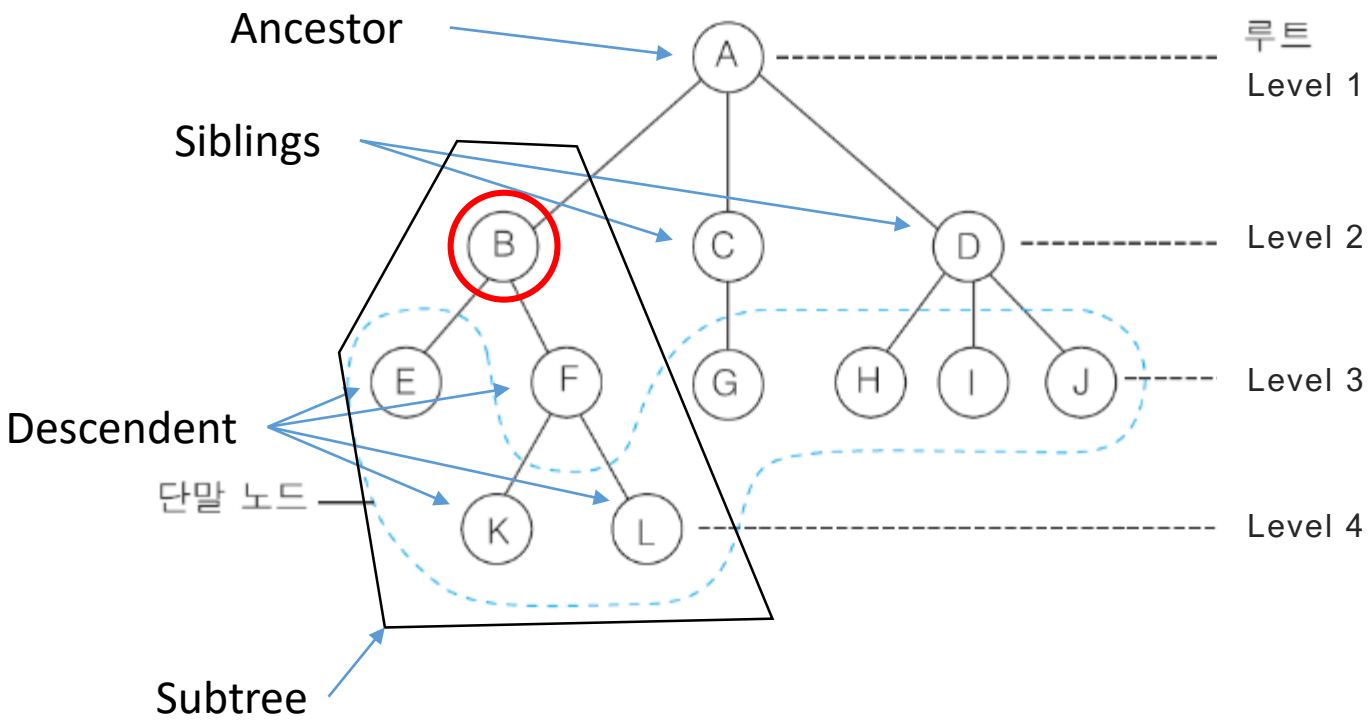
- 트리 T = 부모 – 자식 관계로 이루어진 노드의 집합
- 특징
  - 트리 T는 공집합이 아니며,
  - 루트 노드란 트리 T에서 부모를 갖지 않는 최상위 노드로,
    트리 T는 반드시 하나의 루트 노드를 갖는다.
  - 루트 노드를 제외한 모든 노드들은 단 하나의 부모 노드를 갖는다.



1 구성 요소

- Node: 트리의 원소
- Root: 트리의 시작 노드
- Siblings: 부모가 같은 노드들 (= 형제 노드)
- Leaf: 자식 노드가 없는 노드 (= External Node, 외부 노드)
- Ancestor: 특정 노드에서 루트 노드까지 이어지는 모든 노드들
- Subtree: 부모 노드와의 연결을 끊고 자신을 루트로 삼았을 때 생성되는 트리
- descendent: 서브 트리의 하위 노드들

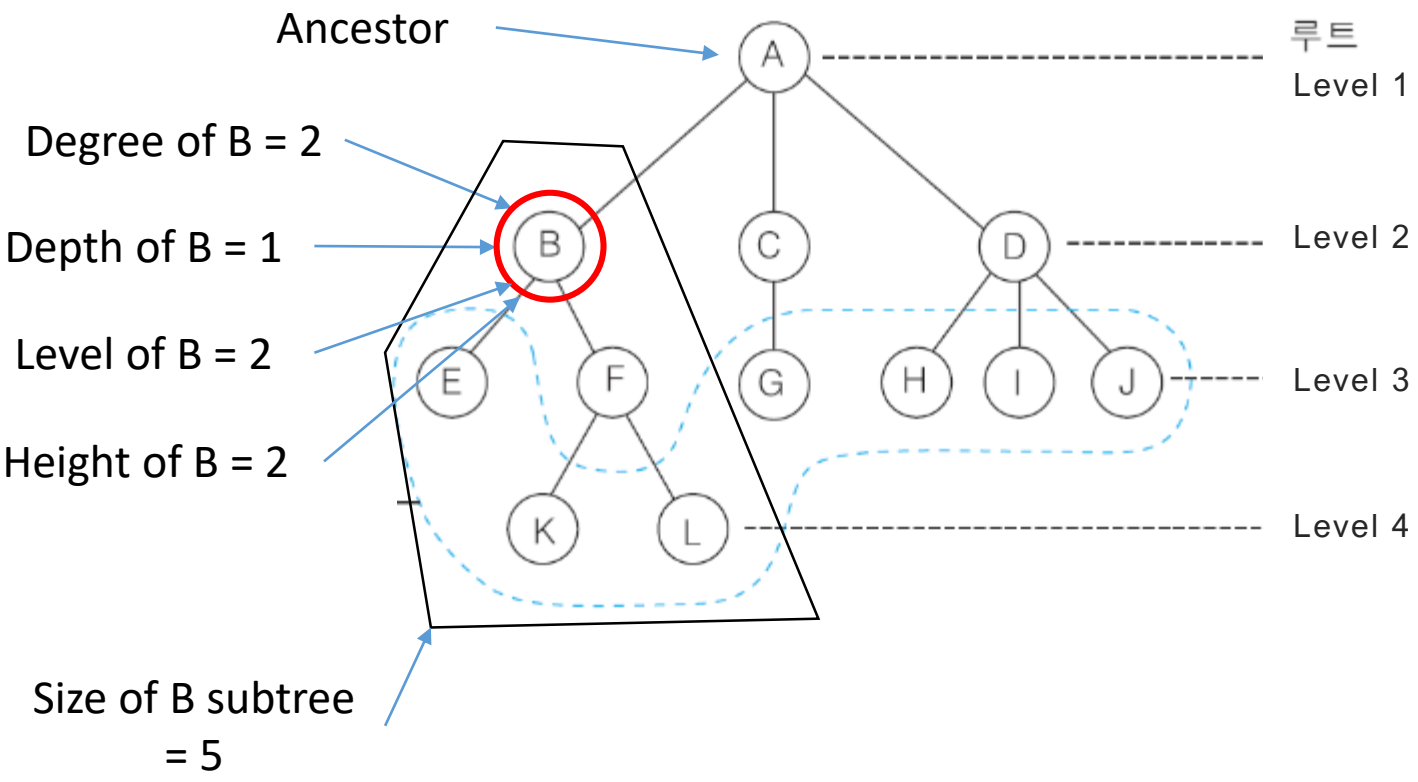
1 구성 요소 (기준 : B노드)


Ancestor
Siblings
Descendent
단말 노드
Subtree
A
B
C
D
E
F
G
H
I
J
K
L
루트
Level 1
Level 2
Level 3
Level 4

1 구성 요소

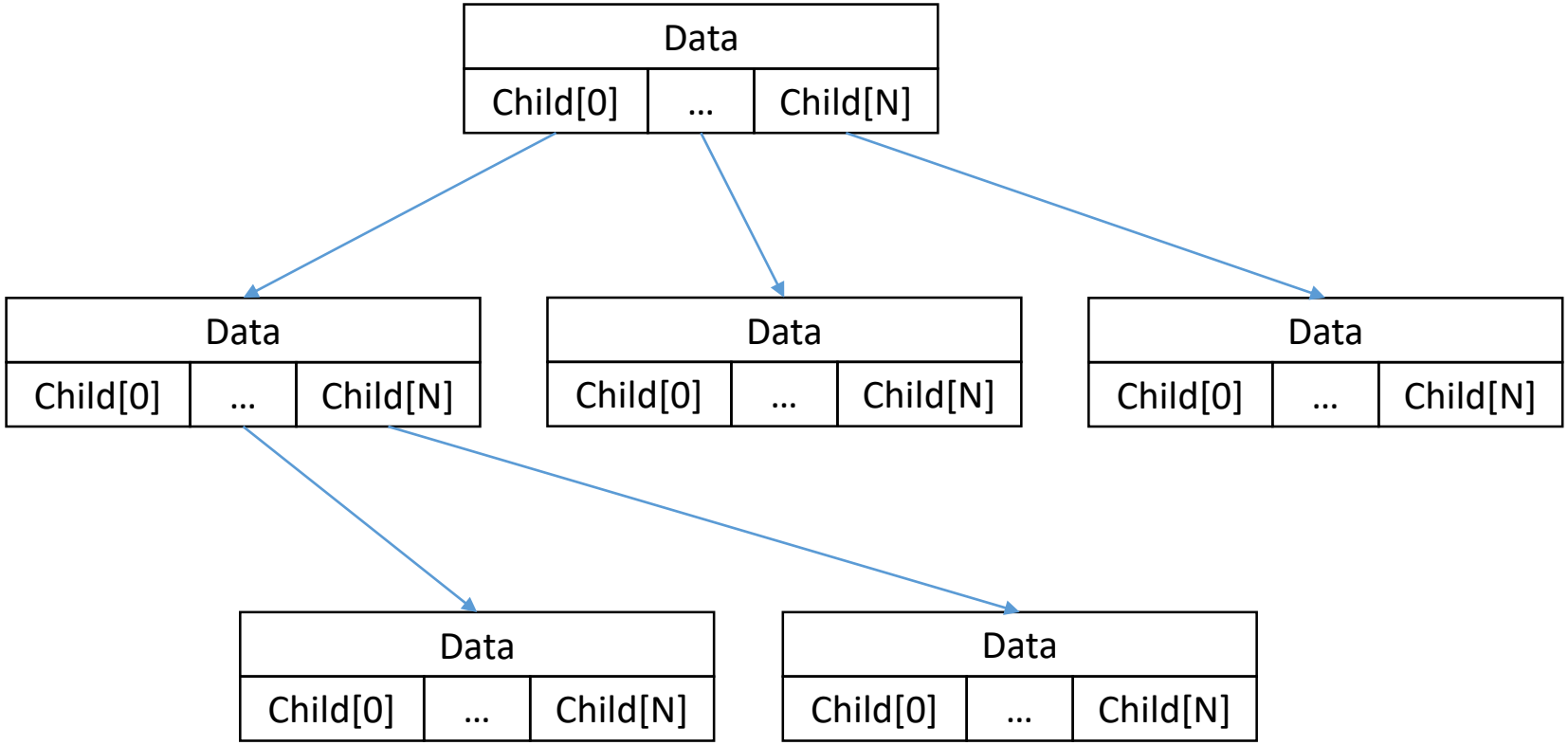
- 노드의 크기(size): 자신을 포함한 모든 자손 노드의 개수
- 노드의 차수(degree): 해당 노드가 가지는 자식의 수
- 노드의 레벨(level): 트리의 특정 깊이를 가지는 노드의 집합
- 노드의 깊이(depth): 루트에서 자신까지 가는 경로의 길이
  - root 노드의 depth는 0
- 노드의 높이(height): 자신과 자손 중 단말 노드 사이의 경로의 최대 길이
  - leaf 노드의 height는 0
- 트리의 깊이(depth) 혹은 높이(height)는 노드들의 깊이 혹은 높이 중 최대값으로 정의된다

1 구성 요소 (기준 : B노드)


Ancestor
Degree of B = 2
Depth of B = 1
Level of B = 2
Height of B = 2
Size of B subtree
= 5
A
B
C
D
E
F
G
H
I
J
K
L
루트
Level 1
Level 2
Level 3
Level 4

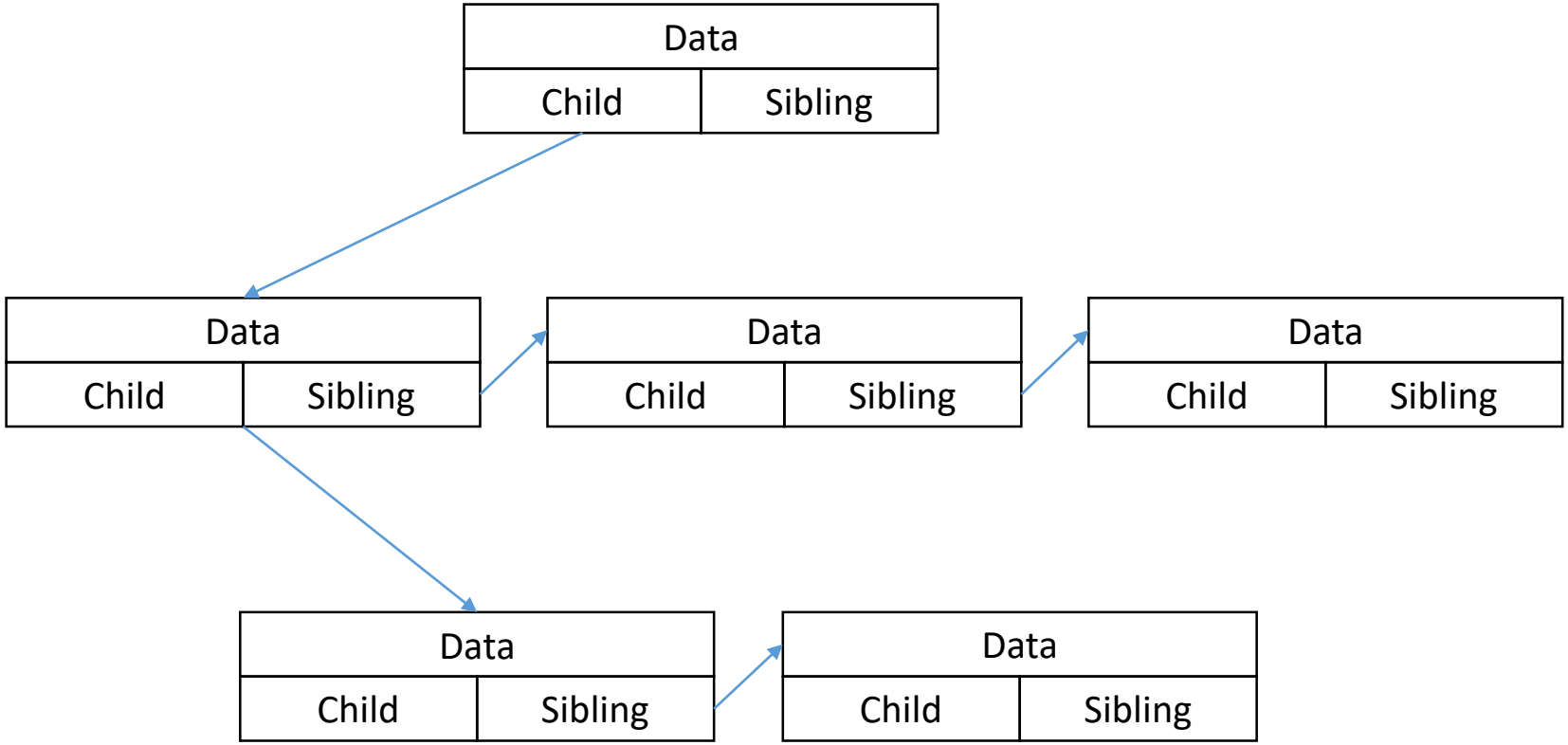
2 일반 트리 표현 방식

N 링크 방법



2 일반 트리 표현 방식

Left Child – Right Sibling 방법



2 일반 트리 표현 방식

- N 링크 방식이 원래 트리의 정의에 부합하는 표현 방식
- 하지만 자식을 무한히 가져야 하는 경우, N 링크로 표현방식은 구현이 복잡할 수 있다 (동적배열 등을 사용하여야 함)
- Left Child – Right Sibling(LCRS) 의 경우, 직관적이지 않지만 구현하는데 있어 N 링크 방식에 비해 좀 더 편리하다
- 하지만 LCRS 로 구현된 트리에서는 기본 연산(탐색, 삽입, 삭제)이 좀 더 오래 걸리는 경우가 많다
  (순회를 하기 위해, 불필요한 형제 노드를 거쳐야함)
- 이번 실습에서는 N 링크 구현법에 대해서만 설명

3 트리 구현을 위한 도구 - 벡터 살펴보기

1 벡터란?

- 벡터: 자동으로 배열의 크기 조절, 객체의 추가와 삭제가 가능한 **동적 배열 구조**
- C++ 표준 템플릿 라이브러리(Standard Template Library: STL)중 하나이다.
- 배열과 사용법이 거의 유사하며, 멤버함수를 통해 다양한 기능을 많이 제공한다.

3 벡터 살펴보기

2 벡터 멤버 함수

벡터는 다음과 같이 선언한다.

```
#include <vector>

using namespace std;
int main() {

vector<int> v;  // < > 안에 벡터의 타입을 명시해야 한다.

return 0;
};
```

3 벡터 살펴보기

2 벡터 멤버 함수

- 벡터를 생성 후, push_back() 함수를 이용하여 언제든지 벡터에 원소를 추가할 수 있다. 이 때 새로운 원소는 벡터의 가장 뒤에 삽입된다.

```
#include <vector>

using namespace std;
int main() {

vector<int> v;
v.push_back(1);
v.push_back(2); // v = {1,2}

return 0;
};
```

3 벡터 살펴보기

2 벡터 멤버 함수

- 벡터의 원소에 대한 접근은 대괄호[]를 이용하면 된다. 이는 배열과 동일하다.

```
#include <vector>

using namespace std;
int main() {

vector<int> v;
v.push_back(1);
v.push_back(2);  // v = {1,2}

int a = v[1];  // a는 2

return 0;
};
```

3 벡터 살펴보기

2 벡터 멤버 함수

- size() 함수를 통해, 현재 벡터안에 존재하는 원소의 개수를 알 수 있다.

```
#include <vector>

using namespace std;
int main() {

vector<int> v;
v.push_back(1);
v.push_back(2);  // v = {1,2}

cout << v.size(); // 현재 v의 길이 2가 출력됨

return 0;
};
```

3 벡터 살펴보기

2 벡터 멤버 함수

- erase() 함수를 통해, 현재 벡터안에 존재하는 원소를 삭제할 수 있다.

```
#include <vector>

using namespace std;
int main() {

vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(3);  // v = {1,2,3}

v.erase(v.begin() + 1);      // v[1]의 원소가 삭제되며, 그 뒤의 값들이 한 칸씩 당겨지게 된다
                             // v = {1,3}
return 0;
};
```

3 벡터 살펴보기

3 벡터 응용

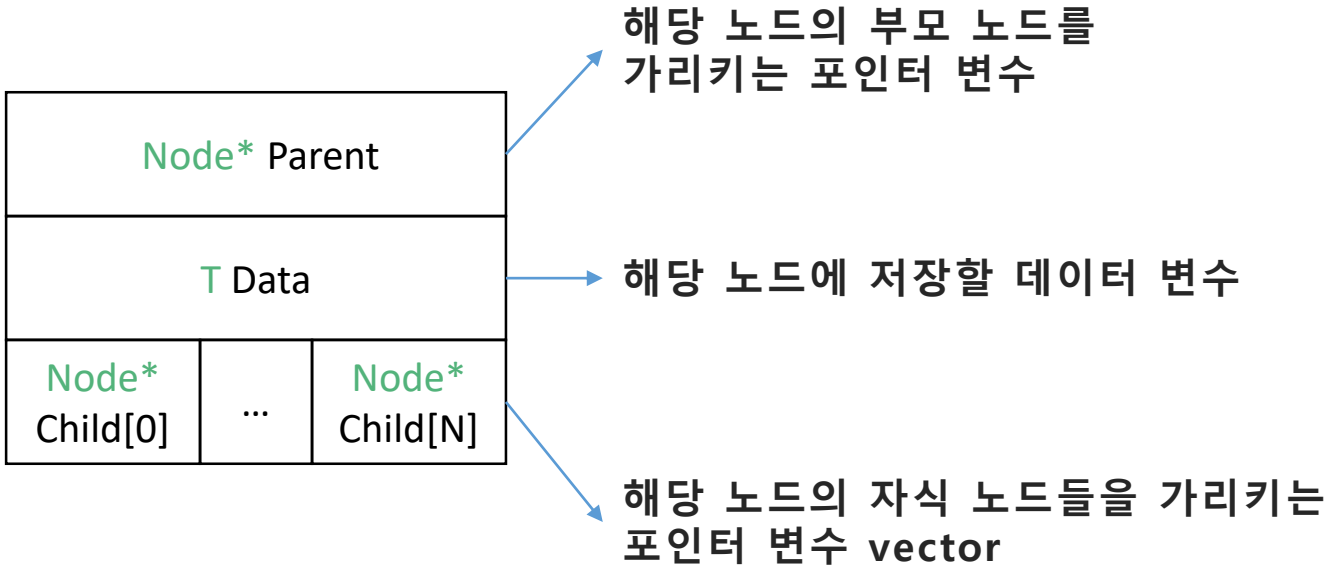
벡터를 사용한 for문과 range 기반 for 문

```
#include <vector>

using namespace std;
int main() {

vector<int> v;
v.push_back(1);
v.push_back(2);
v.push_back(5);

for (int i = 0; i < v.size(); i++) {  // 1 2 5 가 출력됨
cout << v[i] << endl;
}

for (int i : v) {  // 위와 동일한 코드, 1 2 5 가 출력됨
cout << i << endl;
}

return 0;
};
```

4 일반 트리 구현 상세 살펴보기

1 트리 생성

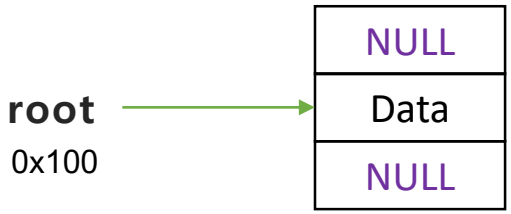
Node 클래스를 정의한다



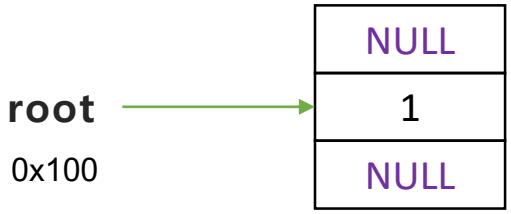
4 일반 트리 구현 상세 살펴보기

1 트리 생성

- root 노드를 하나 가지는 트리를 생성한다



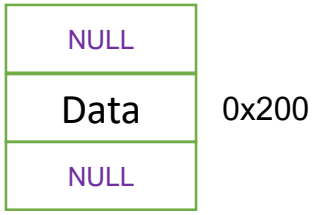
예시) 노드 1을 루트로 하는 트리



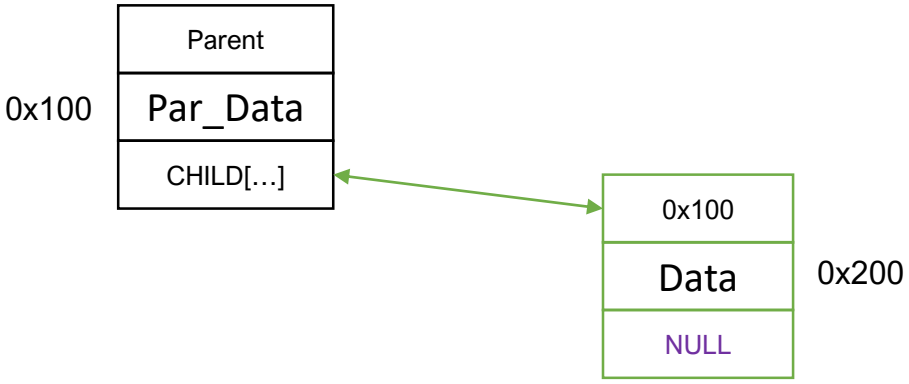
4 일반 트리 구현 상세 살펴보기

2 트리에 노드 삽입

1. 빈 노드를 하나 만든 후 데이터를 저장한다



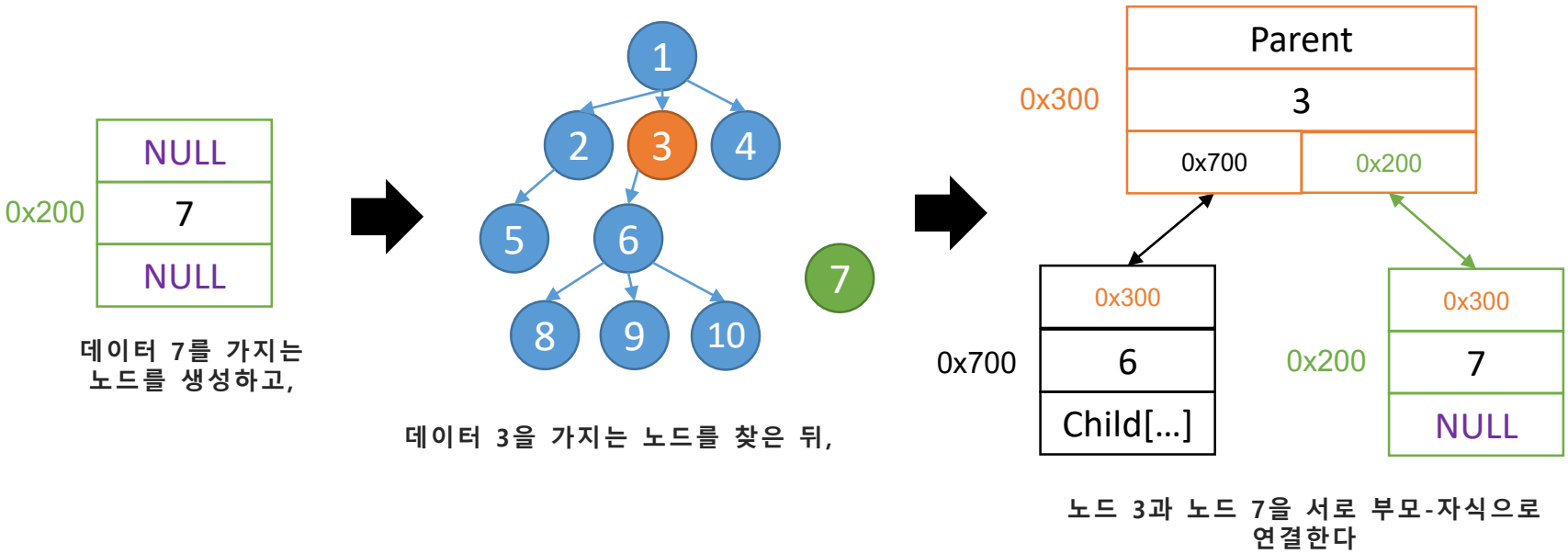
2. 이후 특정 부모 노드를 찾아 그 노드와 부모-자식관계로 연결한다



4 일반 트리 구현 상세 살펴보기

2 트리에 노드 삽입

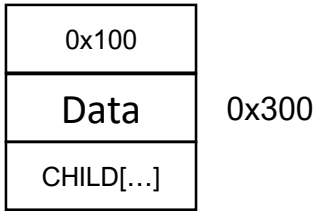
예시) 노드 3의 자식으로 노드 7을 삽입



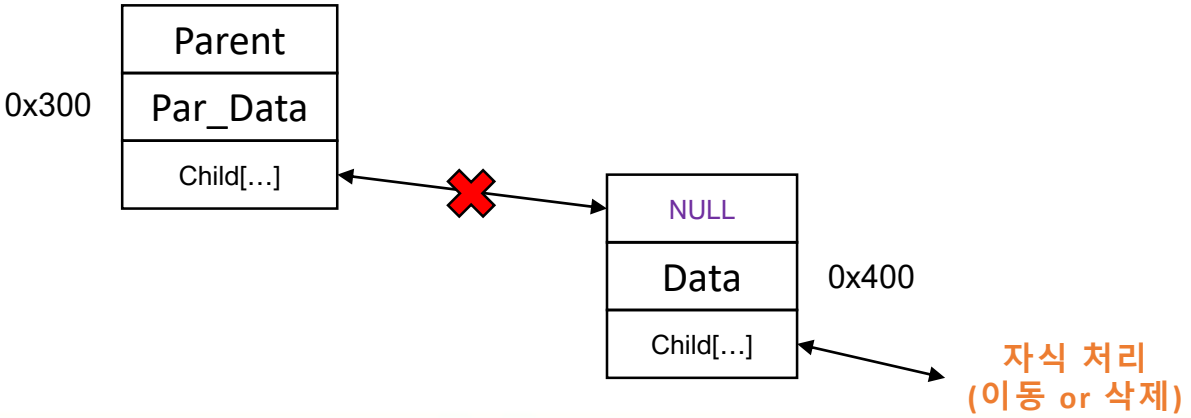
4 일반 트리 구현 상세 살펴보기

3 트리의 노드 삭제

1. 삭제할 노드를 찾는다



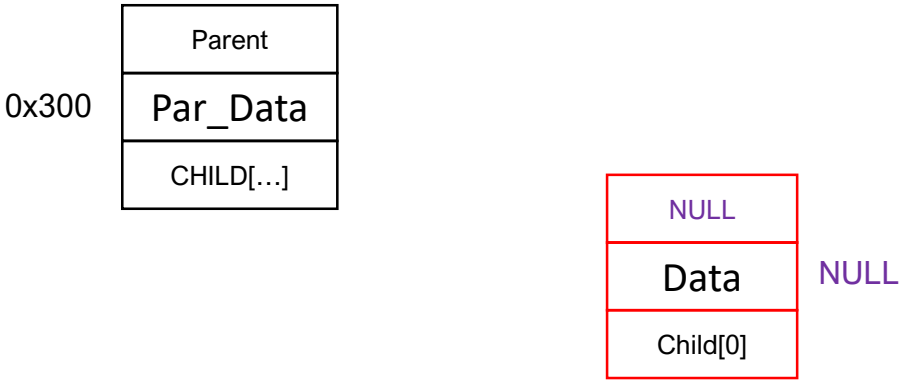
2. 삭제할 노드의 자식들을 적절히 처리하고, 해당 노드의 연결을 끊는다



4 일반 트리 구현 상세 살펴보기

3 트리의 노드 삭제

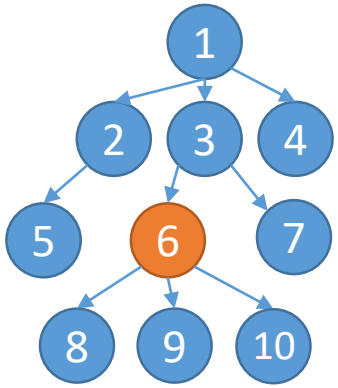
- 해당 노드를 삭제한다.

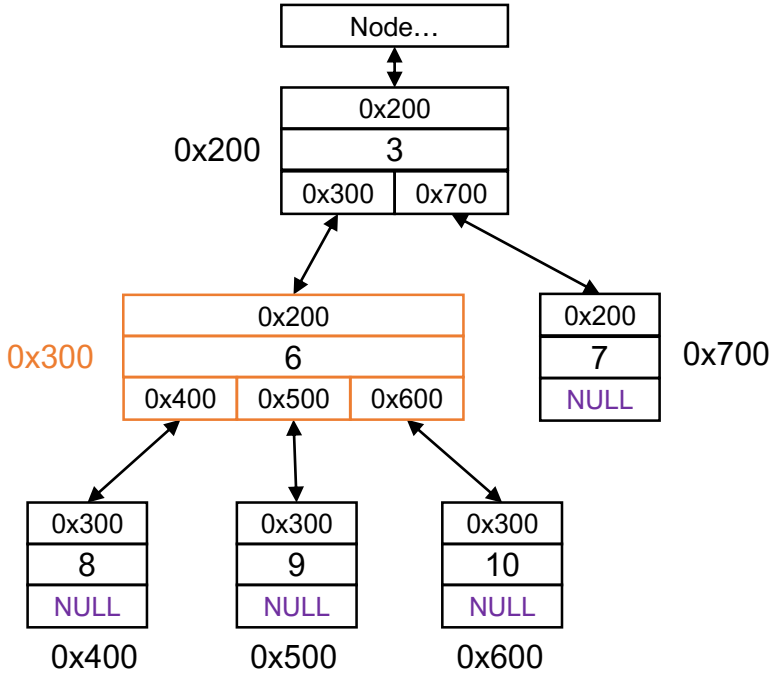


4 일반 트리 구현 상세 살펴보기

3 트리의 노드 삭제

예시) 노드 6을 삭제 (삭제 노드의 자식을 부모 노드의 자식으로 처리)



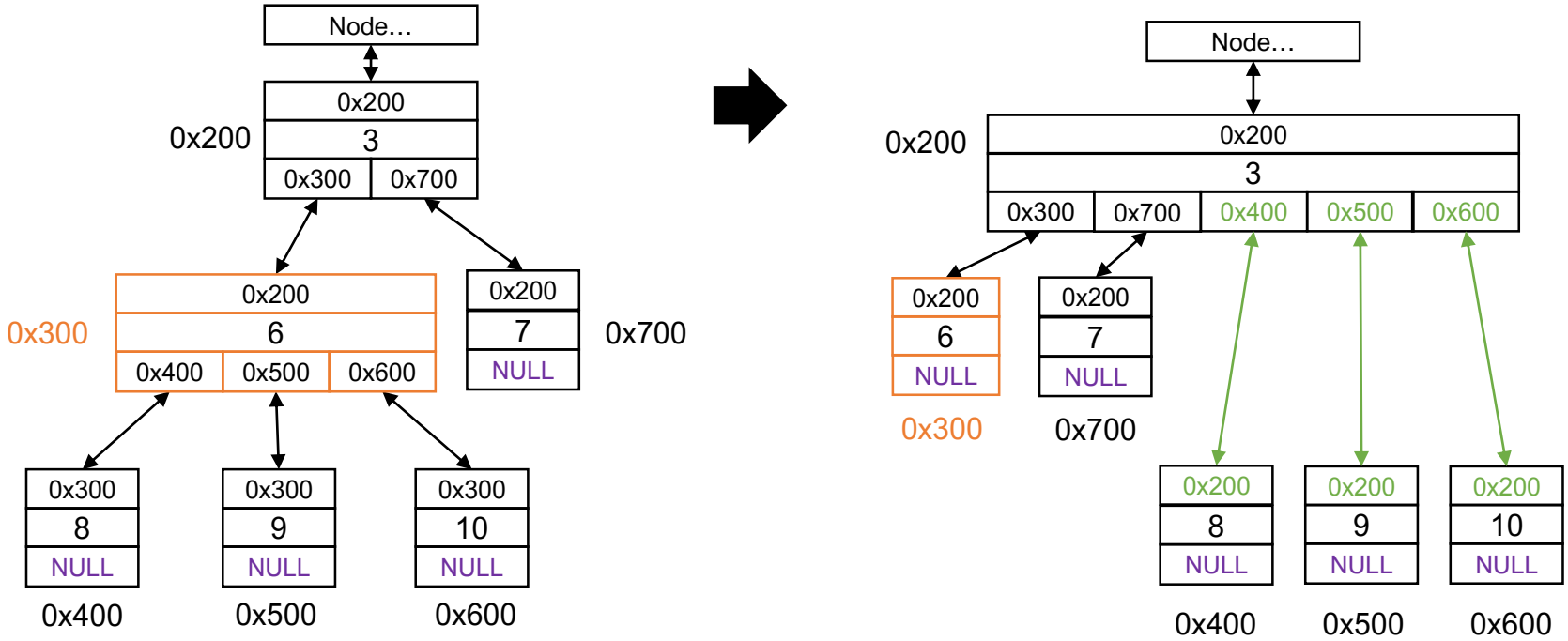


1. 삭제할 데이터 6을 저장하고 있는 노드를 찾는다.

4 일반 트리 구현 상세 살펴보기

3 트리의 노드 삭제

예시) 노드 6을 삭제 (삭제 노드의 자식을 부모 노드의 자식으로 처리)

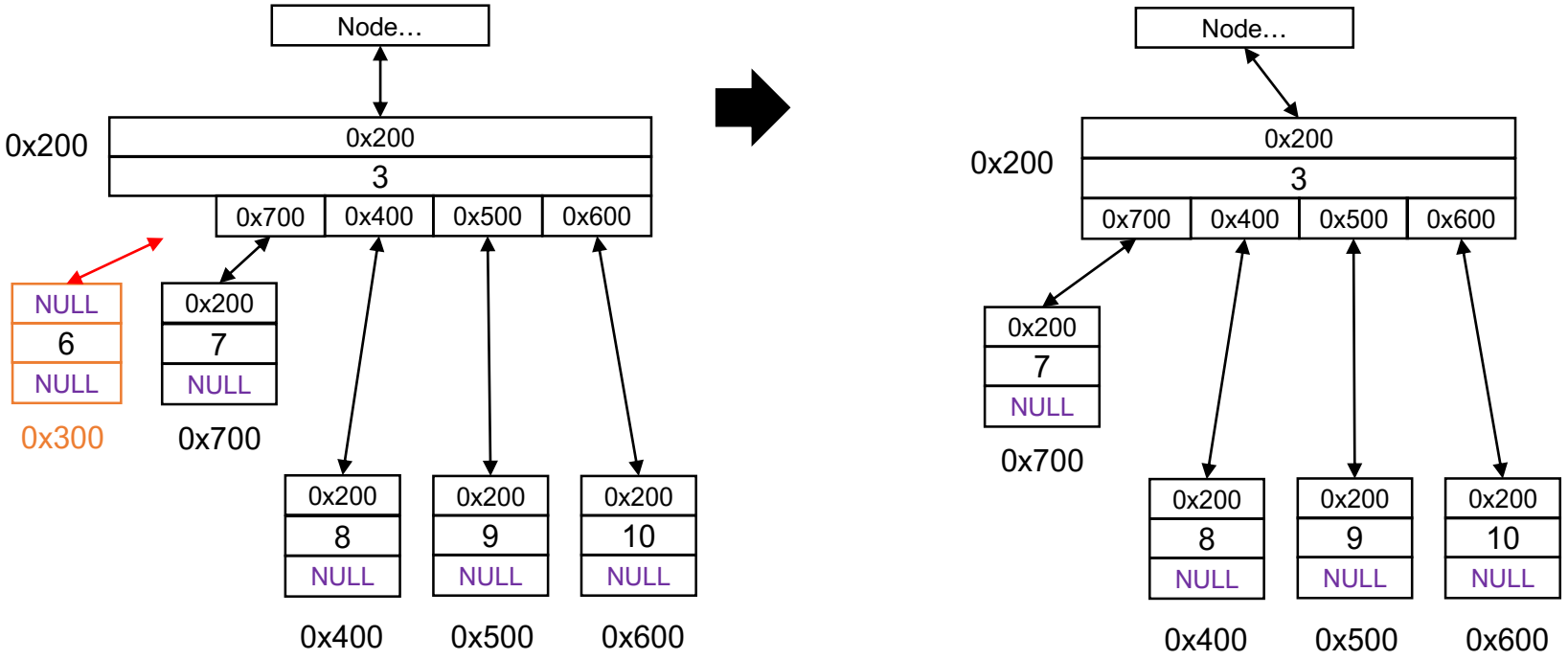


2. 노드 6의 자식 노드들을 6의 부모 노드인 노드 3과 부모-자식 관계로 연결한다

4 일반 트리 구현 상세 살펴보기

3 트리의 노드 삭제

예시) 노드 6을 삭제 (삭제 노드의 자식을 부모 노드의 자식으로 처리)



3. 부모 노드인 3과 자식 노드인 6의 연결을 끊은 후, 노드 6을 삭제한다

5 트리 순회 살펴 보기

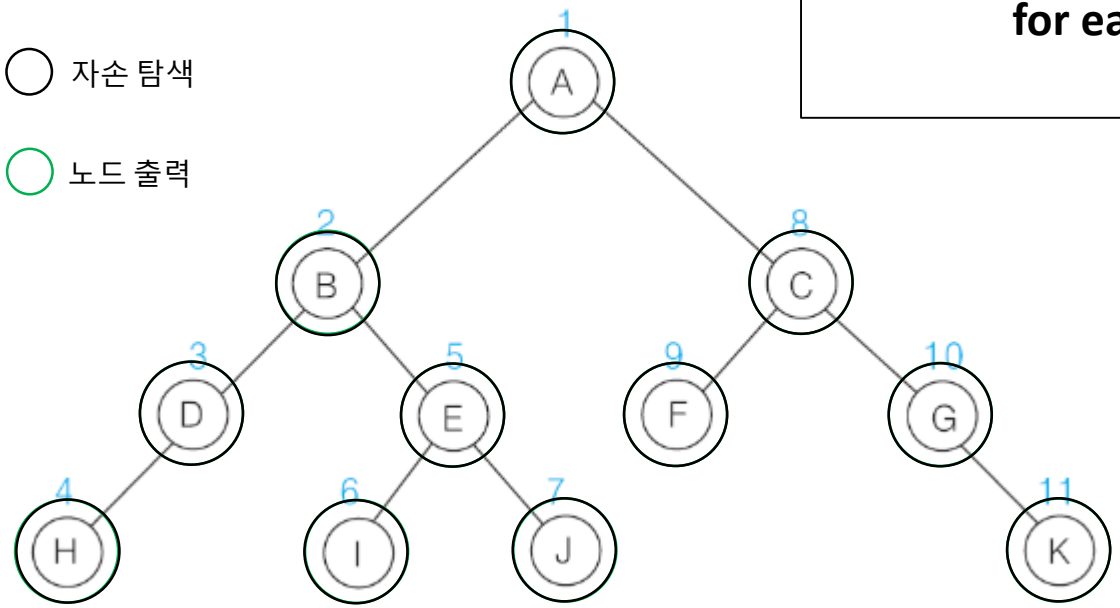
- 재귀(Recursion) : 자신을 정의할 때 자기 자신을 재참조하는 방법
  - 재귀 함수(Recursion Function)는 함수를 정의할 때 자기 자신이 포함됨

```
main()
        recursive(v);
        return 0;

recursive(v)
        print(v);
        recursive(v);
        return;

recursive(v)
        print(v);
        recursive(v);
        return;

recursive(v)
        forever...
```

5 트리 순회 살펴보기 - 전위순회

- 자신을 방문한 후, 자식 노드를 순차적으로 방문
  - 루트를 첫 번째로 방문 후, 자식 노드를 루트로 하는 서브 트리를 재귀적으로 순회

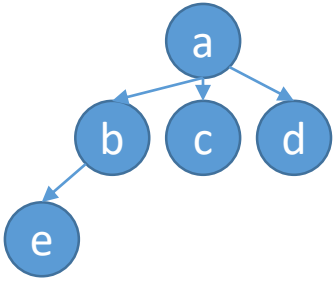
```
Algorithm preOrder(v)
        visit(v)
        for each child w of v:
                preorder (w)
```



5 트리 순회 살펴보기 - 전위순회

출력

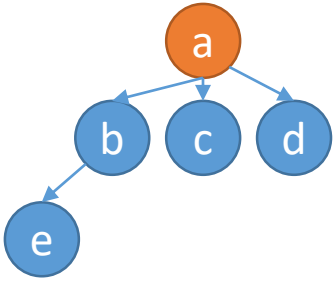
```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```



5 트리 순회 살펴보기 - 전위순회

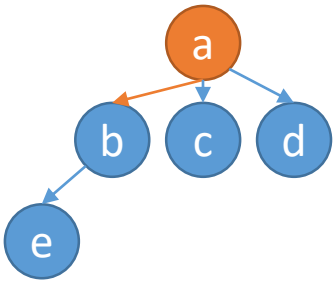
출력
a

```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```



5 트리 순회 살펴보기 - 전위순회

출력
a

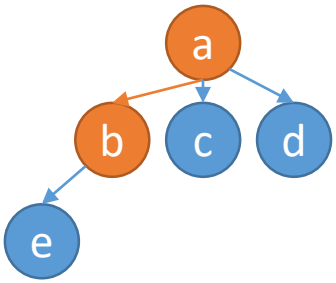



```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```

```
Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;
```

5 트리 순회 살펴보기 - 전위순회

출력
a b

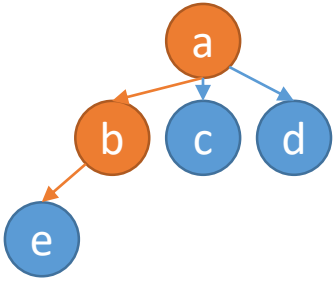



```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```

```
Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;
```

5 트리 순회 살펴보기 - 전위순회

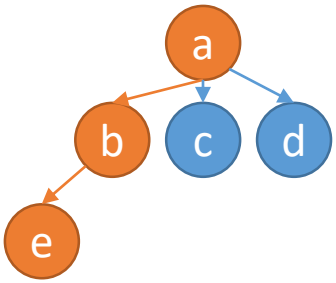
출력
a b




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;

Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;

Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

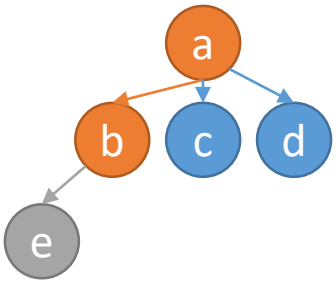
출력
a b e




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;

Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;

Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

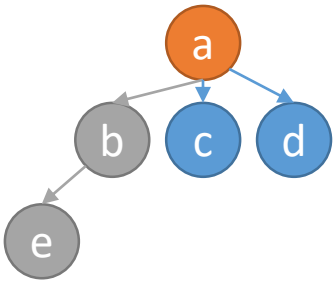
출력
a b e




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;

Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;

Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

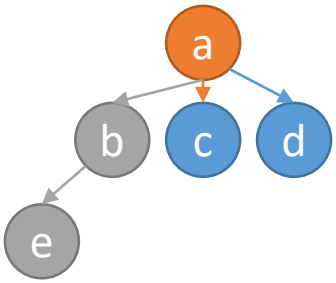
출력
a b e




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;

Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;

Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

출력
a b e




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```

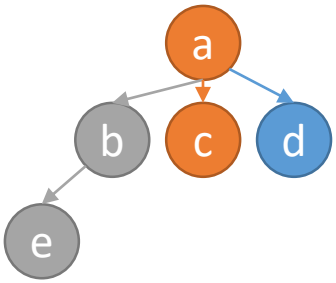
```
Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;
```

```
Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

```
Algorithm preOrder(c)
    visit(c)
    for each child of c:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

출력
a b e c




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```

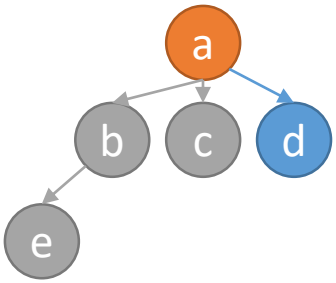
```
Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;
```

```
Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

```
Algorithm preOrder(c)
    visit(c)
    for each child of c:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

출력
a b e c




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```

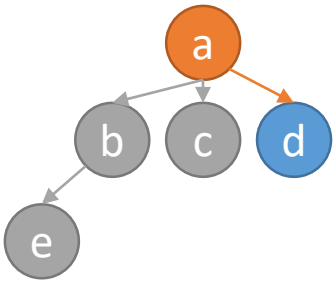
```
Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;
```

```
Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

```
Algorithm preOrder(c)
    visit(c)
    for each child of c:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

출력
a b e c




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```

```
Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;
```

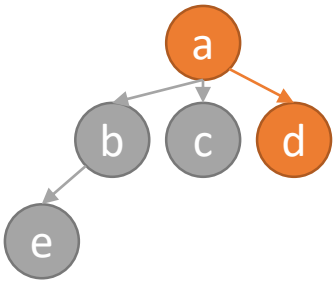
```
Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

```
Algorithm preOrder(c)
    visit(c)
    for each child of c:
        preorder ()
    return;
```

```
Algorithm preOrder(d)
    visit(d)
    for each child of d:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

출력
a b e c d




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```

```
Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;
```

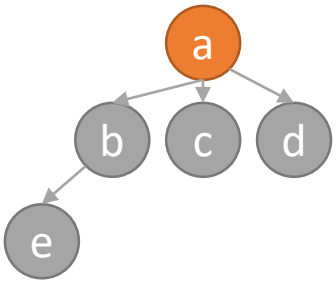
```
Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

```
Algorithm preOrder(c)
    visit(c)
    for each child of c:
        preorder ()
    return;
```

```
Algorithm preOrder(d)
    visit(d)
    for each child of d:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

출력
a b e c d




```
Algorithm preOrder(a)
    visit(a)
    for each child of a:
        preorder (child)
    return;
```

```
Algorithm preOrder(b)
    visit(b)
    for each child of b:
        preorder (child)
    return;
```

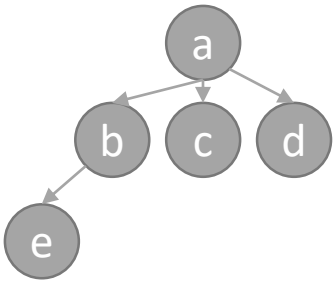
```
Algorithm preOrder(e)
    visit(e)
    for each child of e:
        preorder ()
    return;
```

```
Algorithm preOrder(c)
    visit(c)
    for each child of c:
        preorder ()
    return;
```

```
Algorithm preOrder(d)
    visit(d)
    for each child of d:
        preorder ()
    return;
```

5 트리 순회 살펴보기 - 전위순회

출력
a b e c d



Algorithm *preOrder*(*a*)
visit(a)
for each *child* of *a*:
preorder (*child*)
return;

Algorithm *preOrder*(*b*)
visit(b)
for each *child* of *b*:
preorder (*child*)
return;

Algorithm *preOrder*(*e*)
visit(e)
for each *child* of *e*:
preorder ()
return;

Algorithm *preOrder*(*c*)
visit(c)
for each *child* of *c*:
preorder ()
return;

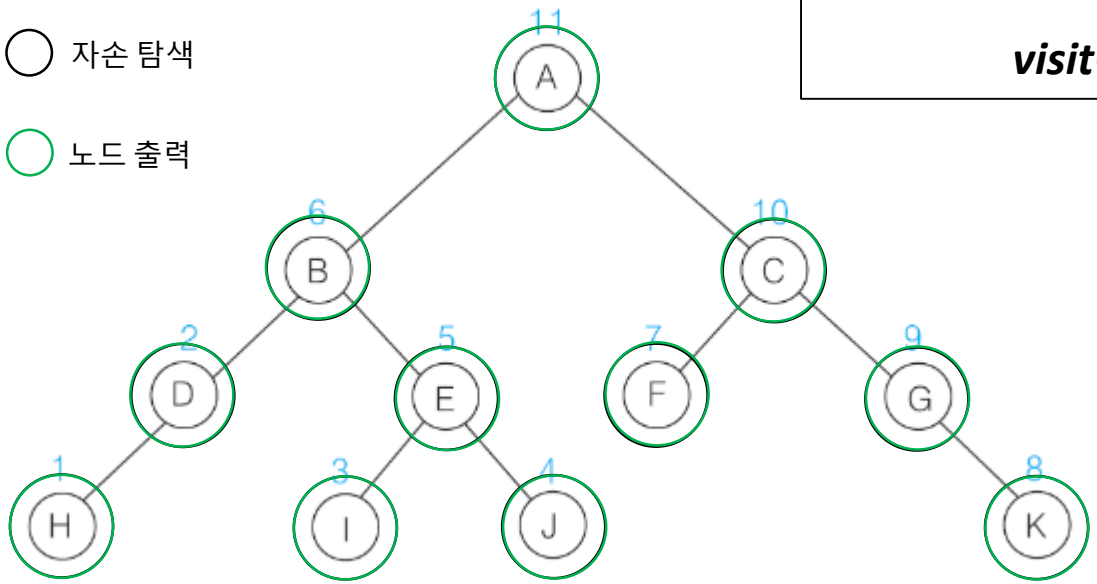
Algorithm *preOrder*(*d*)
visit(d)
for each *child* of *d*:
preorder ()
return;

5 트리 순회 살펴보기 - 후위순회

자식 노드를 순차적으로 모두 방문한 후, 자신을 방문

- 루트의 자식 노드를 루트로 하는 서브 트리를 먼저 재귀적으로 순회한 후 루트를 방문

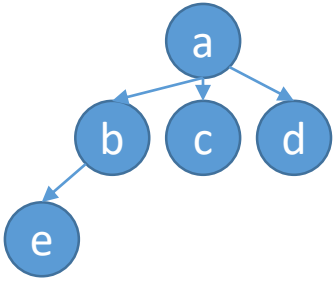
```
Algorithm postOrder(v)
        for each child w of v:
                postorder (w)
        visit(v)
```



5 트리 순회 살펴보기 - 후위순회

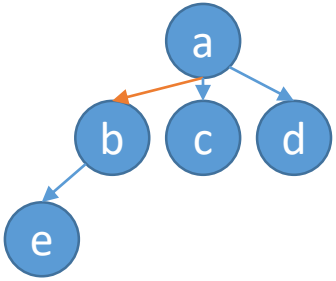
출력

```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;
```



5 트리 순회 살펴보기 - 후위순회

출력

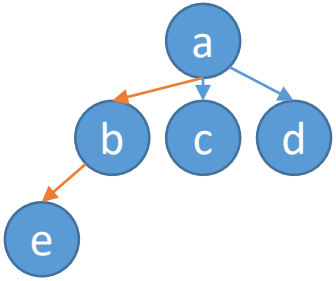



```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;
```

```
Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;
```

5 트리 순회 살펴보기 - 후위순회

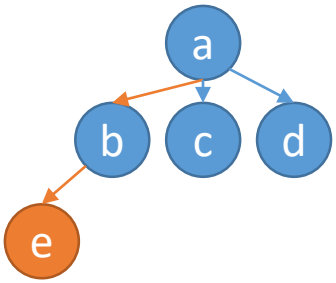
출력




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;

Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;

Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(b)
    return;
```

5 트리 순회 살펴보기 - 후위순회

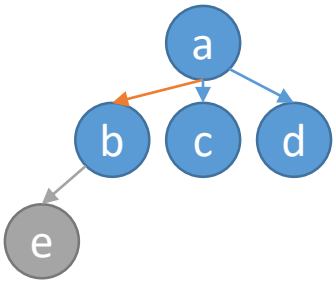
출력
e




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;

Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;

Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;
```

5 트리 순회 살펴보기 - 후위순회

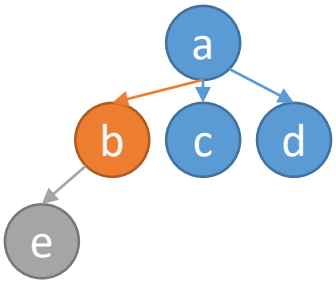
출력
e




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;

Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;

Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;
```

5 트리 순회 살펴보기 - 후위순회

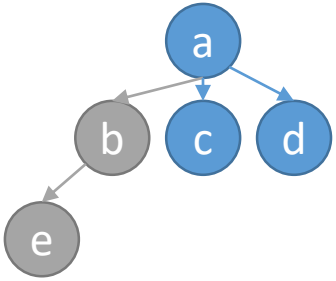
출력
e b




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;

Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;

Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;
```

5 트리 순회 살펴보기 - 후위순회

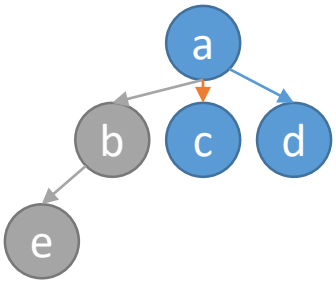
출력
e b




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;

Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;

Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;
```

5 트리 순회 살펴보기 - 후위순회

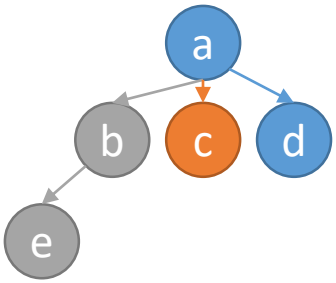
출력
e b




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;

Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;

Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;

Algorithm postOrder(c)
    for each child of c:
        postorder ()
    visit(c)
    return;
```

5 트리 순회 살펴보기 - 후위순회

출력
e b c




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;
```

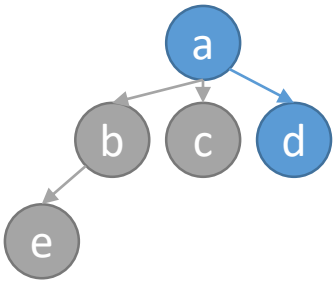
```
Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;
```

```
Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;
```

```
Algorithm postOrder(c)
    for each child of c:
        postorder ()
    visit(c)
    return;
```

5 트리 순회 살펴보기 - 후위순회

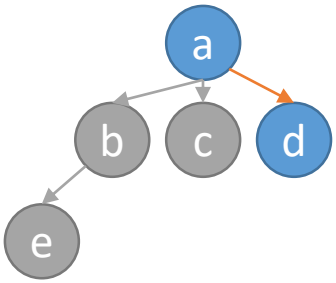
출력
e b c




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;

Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;

Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;

Algorithm postOrder(c)
    for each child of c:
        postorder ()
    visit(c)
    return;
```

5 트리 순회 살펴보기 - 후위순회

출력
e b c




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;
```

```
Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;
```

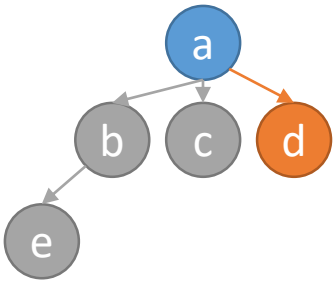
```
Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;
```

```
Algorithm postOrder(c)
    for each child of c:
        postorder ()
    visit(c)
    return;
```

```
Algorithm postOrder(d)
    for each child of d:
        postorder ()
    visit(d)
    return;
```

5 트리 순회 살펴보기 - 후위순회

출력
e b c d




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;
```

```
Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;
```

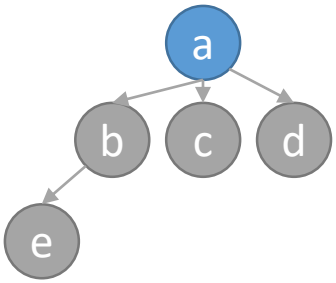
```
Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;
```

```
Algorithm postOrder(c)
    for each child of c:
        postorder ()
    visit(c)
    return;
```

```
Algorithm postOrder(d)
    for each child of d:
        postorder ()
    visit(d)
    return;
```

5 트리 순회 살펴보기 - 후위순회

출력
e b c d




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;
```

```
Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;
```

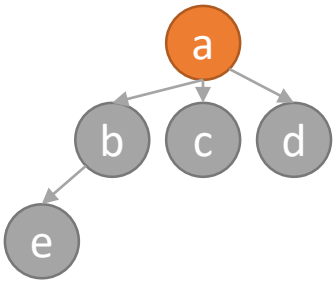
```
Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;
```

```
Algorithm postOrder(c)
    for each child of c:
        postorder ()
    visit(c)
    return;
```

```
Algorithm postOrder(d)
    for each child of d:
        postorder ()
    visit(d)
    return;
```

5 트리 순회 살펴보기 - 후위순회

출력
e b c d a




```
Algorithm postOrder(a)
    for each child of a:
        postorder (child)
    visit(a)
    return;
```

```
Algorithm postOrder(b)
    for each child of b:
        postorder (child)
    visit(b)
    return;
```

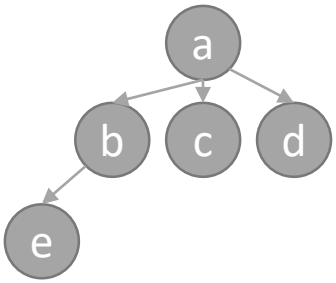
```
Algorithm postOrder(c)
    for each child of c:
        postorder ()
    visit(c)
    return;
```

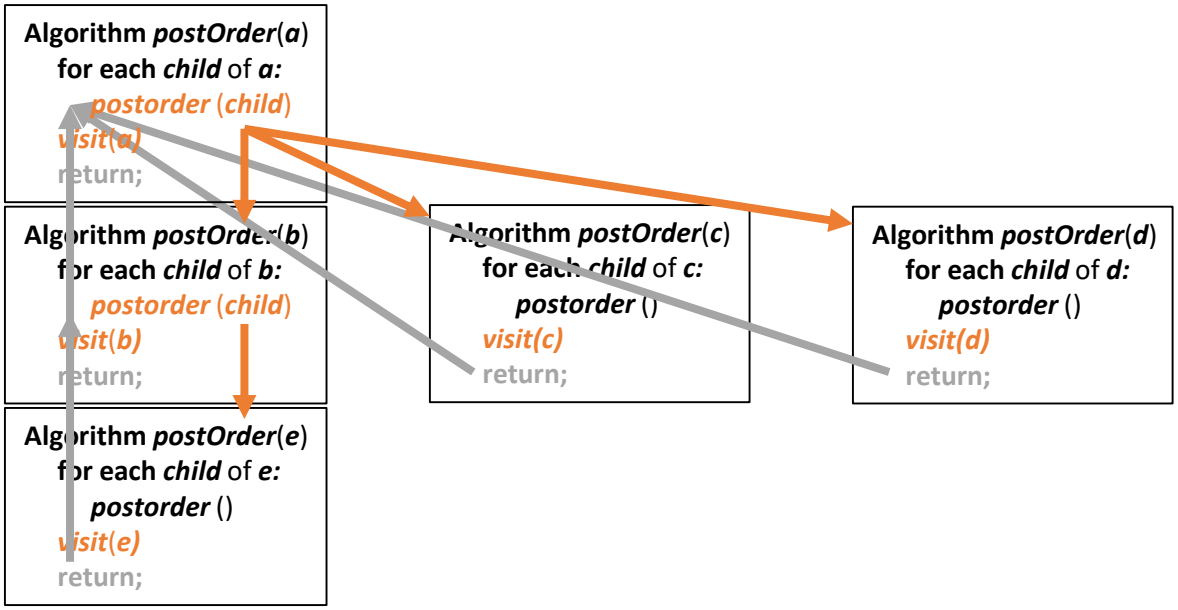
```
Algorithm postOrder(d)
    for each child of d:
        postorder ()
    visit(d)
    return;
```

```
Algorithm postOrder(e)
    for each child of e:
        postorder ()
    visit(e)
    return;
```

5 트리 순회 살펴보기 - 후위순회

출력
e b c d a





6 기본 코드 템플릿

```cpp
#include <vector>
class Node {
public:
    int data;
    Node* par;
    std::vector<Node*> chi;

    Node() {
        this->data = NULL;
        this->par = NULL;
    }
    Node(int data) {
        this->data = data;
        this->par = NULL;
    }
    ~Node() {
    }
    void insertChild(Node* chi) {    // add child node to this node
        this->chi.push_back(chi);
        return;
    }
    void delChild(Node* chi) {  // delete child node of this node
        for (int i = 0; i < this->chi.size(); i++) {
            if (this->chi[i] == chi) {
                this->chi.erase(this->chi.begin() + i);
                break;
            }
        }
        return;
    }
};
```

6 기본 코드 템플릿

```
53
54  class GeneralTree {
55  private:
56      Node* root;
57      std::vector<Node*> node_list; // list saving all node pointer in Tree
58  public:
59      GeneralTree() {
60          root = NULL;
61      }
62      GeneralTree(int data) {
63          root = new Node(data);
64          node_list.push_back(root);
65      }
66      ~GeneralTree() {
67      }
68      void setRoot(int data) {
69          root = new Node(data);
70          node_list.push_back(root);
71      }
72      Node* getRoot() {
73          return root;
74      }
75      void insertNode(int parent_data, int data){ ... }
93
94      void delNode(int data){ ... }
116
117     Node* findNode(int data){ ... }
120
121     void printChild(int data){ ... }
135
136     int countdepth(int data){ ... }
151
152     void preorder(Node* node){ ... }
159
160     void postorder(Node* node){ ... }
167 };
168
169
```


Thank you